TEXAS A&M UNIVERSITY

Hardware Implementation of a Desktop Supercomputer for High Performance Image Processing

ONR Grant Number N00014-94-1-0516

Technical Report
May/01/94 – Aug/01/94

DEPARTMENT OF ELECTRICAL ENGINEERING
College Station, Texas

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF COLOR PAGES WHICH DO NOT REPRODUCE LEGIBLY ON BLACK AND WHITE MICROFICHE.

Hardware Implementation of a Desktop Supercomputer for High Performance Image Processing

ONR Grant Number N00014-94-1-0516

Technical Report
May/01/94 – Aug/01/94

COLOR IMAGE PROCESSING USING CELLULAR NEURAL NETWORKS

Dr. Jose Pineda de Gyvez

Texas A&M University
Microelectronics Group
Department of Electrical Engineering
College Station, TX, 77843

Phone: (409) 8457477
FAX: (409) 8457161
Email:gyvez@pineda.tamu.edu

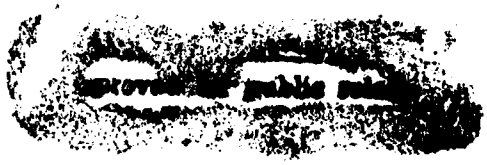
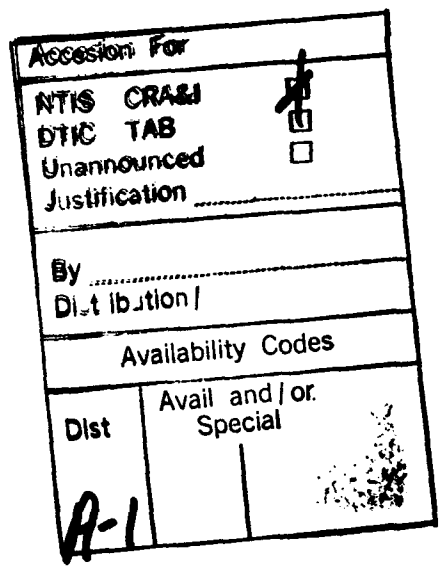


TABLE OF CONTENTS



1. INTRODUCTION

The *Cellular Neural Network* (CNN) paradigm has rapidly evolved to cover a wide range of applications which are typically characterized by their spatial dynamics[1]. One particular area of big interest is filtering for image processing. Enormous advances have been made by many researchers in this field[2]. Among the relevant work is the creation of templates capable of several image processing applications and the development of special purpose algorithms such as halftoning and character recognition, to mention some[3–6]. However, except for the fundamental work on color processing developed by Roska et. al.[7], all of the work presented so far deals only with black and white images. This probably stems from the fact that the CNN paradigm was created only for binary states, giving up, somehow, the wide range that the activation energy is capable of. Yet, to handle realistic situations it is necessary to advance the state of the art into color image processing.

Another interesting and propitious area of research concerns multi–layer CNN[8]. Simulation strategies based on CNN multi–layer architectures are an ideal vehicle for color image processing. This is because each pixel's color can be handled as a triplet *<Red Green Blue>* whose combinations yield a secondary color. It follows then that it is possible to allocate a layer of CNN cells to each primary color component, carry out the processing independently, and then form the triplet to see the results. We call this approach a *sequential color processing* mode. The counterpart is a *concurrent color processing* mode. This mode results from applications in which it is not desired to split a pixel's color into its three basic components. In this case it would be necessary to have an output function that would be based on the Euclidean distance, or any other norm, of the three *RGB* values.

The basic structure of the simulator presented in this paper is based on a high performance software capable of efficiently dealing with large images in the order of 10^5 pixels [9, 10]. The simulator operates in a *sequential* mode. This provides an added flexibility to create individual templates that can be applied on single colors to obtain a full mix of applications/colors. The simulator runs in an X–Windows environment and uses the Graphics Interface Format (GIF) format as standard input.

Although CNN is already well established in the literature, its link to cellular automata has only been sketched. Therefore, as preliminary background, section 2 introduces the CNN paradigm based on strict definitions of automata networks. Sections 3 and 4 address the color processing capabilities and behavioral simulation approach of our software. Section 5 presents the software environment and post–processing capabilities for image processing. Section 6 acquaints us with some image processing applications of CNN in practical situations and also presents comparisons between different color–mapping strategies implemented in the simulator.

2. BACKGROUND THEORY

Cellular automata was originally introduced by Ulam and von Neumann as a particular case of Automata Networks[11]. Cellular Automata is distinguished mainly because its graph follows a regular lattice Z. Its neighborhood structure and the transition function among vertices are translation invariant, i.e. they are the same for all vertices; additionally, the state updating rule is synchronous.

Let $I = Z^d$, where d is the dimension of the lattice. If a set of connections $V \subset Z^d \times Z^d$ is translation invariant, meaning $(j, i) \in V$ iff $(j+k, i+k) \in V$, the graph $G = (Z^d, V)$ is called a cellular space. Cellular Automata are automata defined on the cellular space whose transition function is also translation invariant: $f_i = f$ for any $i \in Z^d$ with $f : Q^{|V|} \rightarrow Q$. and Q the set of states.

An important class of Automata Networks are the McCulloch–Pitts Automata, also called *Neural Networks*. Its graph $G = (I, V)$ possesses a *weighted* structure: for every edge $(j, i) \in V$, a real number $w_{ij} \in R$ is assigned to it to represent its weight. The state set is usually $Q = \{-1, 1\}$ or $Q = \{0, 1\}$. When the state set Q is $\{-1, 1\}$, the local functions are as follows: $f_i : \{-1, 1\}^{|V_i|} \rightarrow \{0, 1\}$, $f_i(x_i : j \in V_i) = sign\left(\sum_{j \in V_i} w_{ij}\, x_j - b_i\right)$ where *sign* is the *threshold* function (or the *activation* function, as it is known in Neural Network literature). The weights are interpreted as synapses which are either *excitatory* if the weight $w > 0$, or *inhibitory* if $w < 0$. Therefore, the state of the neuron at vertex i will be excited if the weighted sum $\sum_{j \in V_i} w_{ij}\, x_j$ is greater than its threshold b_i. A *bias* can be included by adding a component $x_0 = 1$ to the vector x, and it is treated exactly like any other weight. The well known *bipolar sigmoid* function with range from -1 to $+1$ is often used as the activation function for networks in which the desired outputs values either are -1 or $+1$ or are in the interval between -1 and $+1$. Furthermore, Neural Networks are often classified as single–layer or multi–layer, and the number of layers in the network can be defined to be the number of layers of weighted interconnection links between the neurons. In summary, a Neural Network is characterized by its *architecture*, i.e. its pattern of connections between the neurons, its *training algorithm*, i.e. its method of determining the weights on the connections, and its *activation function*.

CNN possesses some of the key features of both Cellular Automata and Neural Networks. Like Cellular Automata, it has a structure of cellular space, Z^d, with the neighborhood and the transition function being translation invariant. The neighborhood is shown in Fig. 1a. Notice the full interconnection between a cell and its neighboring cells. The transition function, f_i, in this case is a nonlinear differential equation. Both CNN and Cellular Automata have parallel signal processing capability, and their dynamics is based on the nearest neighbor interactions. There are also differences between the two, the main one being that while CNN is a continuous time dynamical system, Cellular Automata is a discrete time system. Like Neural Networks, CNN's continuous time feature allows real–time signal processing. Among the similarities are the concepts of weighted interconnections, w_{ij}, the bias, b, and the activation function. The multi–layer feature is also used in CNN.

2.1. CELLULAR NEURAL NETWORKS

The basic unit of CNN is called a *cell* [12, 13]. Any cell, *C(i,j)*, is connected only to its neighbor cells, i.e. adjacent cells interact directly with each other. This *neighborhood* is denoted as *N(i,j)*. Cells not in the immediate neighborhood have indirect effect because of the propagation effects of the dynamics of the network. Each cell has a state x, input u, and output y. The state of each cell is bounded for all time $t > 0$ and, after the transient has settled down, a cellular neural network always approaches one of its stable equilibrium points. This last fact is relevant because it implies that the system will

not oscillate. The dynamics of a CNN has both output feedback *(A)* and input control *(B)* mechanisms. The equivalent block diagram of a continuos–time cell is shown in Fig. 1b. The first order nonlinear differential equation defining the dynamics of a cellular neural network cell can be written as follows

$$C\frac{dx_{ij}(t)}{dt} = -\frac{1}{R}x_{ij}(t) + \sum_{C(k,l)\in N_r(i,j)} A(i,j;k,l)\ y_{kl}(t) + \sum_{C(k,l)\in N_r(i,j)} B(i,j;k,l)\ u_{kl} + I \qquad (1)$$

$$y_{ij}(t) = \frac{1}{2}(|x_{ij}(t) + 1| - |x_{ij}(t) - 1|) \qquad (2)$$

where x_{ij} is the state of cell $C(i,j)$, $x_{ij}(0)$ is the initial condition of the cell, C and R conform the integration time constant of the system, and I is an independent bias constant. $A(i,j;k,l)\ y_{kl}$ and $B(i,j;k,l)\ u_{kl}$ are programming templates for all cells $C(k,l)$ in the neighborhood $N(i,j)$ of cell $C(i,j)$, and y_{ij} represents the output equation, i.e. the activation function for the cell. This function is shown in Fig. 2.

Notice from the summation operators that each cell is affected by its neighbor cells. $A(\cdot)$ acts on the output of neighboring cells and is referred to as the *feedback operator.* $B(\cdot)$ in turn affects the input control and is referred to as the *control operator.* Specific entry values of matrices $A(\cdot)$ and $B(\cdot)$ are application dependent and space invariant. The matrices are also known as *cloning templates*[14, 15]. A constant bias I and the cloning templates determine the *transient behavior* of the cellular nonlinear network.

In image processing applications the concept of locality is important. Usually, a pixel's value is calculated based only on its neighbor pixels. Neural Networks like Hopfield lack this property of locality making them unsuitable for image processing applications.

3. COLOR PROCESSING

To perform any kind of color image processing, a color model must be selected. With CNN, there is no exception. The purpose of a color model is to facilitate the specification of colors in some standard manner. Basically, a color model is a specification of a 3–dimensional coordinate system and a subspace within that system where each color is represented by a point[16, 17].

In most cases, a pixel's value is indexed into a table of colors referred to as palette. However, bitmaps that represent a large number of colors simultaneously generally do not employ the palette scheme because it is too costly in terms of memory. Virtually, all computer display hardware employs the *RGB (Red Green Blue)* model. In this scheme a color is represented by the relative amounts of color (intensities) of three primary colors that are required to produce the given color. The *RGB* model is an *additive primary system* that describes a color in terms of the percentage of red, green, and blue in the color. These three colors are called *additive primaries.* Mixing them is like combining colored lights: combining 100% red, 100% green, and 100% blue creates white, i.e. <255, 255, 255> in RGB values. Conversely, combining 0% red, 0% green, and 0% blue creates black, <0, 0, 0> in RGB values.

The model chosen for color image processing with CNN is the *RGB* model. Using the RGB model has the advantage that each primary color can be represented by a CNN layer, e.g. red, green and blue layers $\mathcal{L}_R$, $\mathcal{L}_G$, $\mathcal{L}_B$. Thus, a simulation approach is to have the triplet <RGB> processed by a three–layer CNN, with each layer processing a primary color. Following this idea it is then possible to apply distinct templates to each color layer and even to apply templates in between color layers. Therefore, with the ability to process *RGB* separately, plus the interlayer template effects, more complex image processing applications can be done. It is thus possible to do, say, edge detection in $\mathcal{L}_R$, and averaging in $\mathcal{L}_G$, simultaneously.

To be able to work with multiple layers, the basic CNN equation (1) can rapidly be expanded to a matricidal equation of the following form

$$\frac{dx_{ij}(t)}{dt} = -x_{ij}(t) + \sum_{C(k,l)\in N_r(i,j)} \underline{A}(i,j;k,l)\ y_{kl}(t) + \sum_{C(k,l)\in N_r(i,j)} \underline{B}(i,j;k,l)\ u_{kl} + I \qquad (3a)$$

$$y_{ij}(t) = \frac{1}{2}(|x_{ij}(t) + 1| - |x_{ij}(t) - 1|) \qquad (3b)$$

where for simplicity the time integration constant has been assumed to be unity. In this last equation, instead of only one state variable per cell there are three state variables to be able to process color. $\underline{A}$ and $\underline{B}$ are block triangular matrices and **I, x, y** are vectors as follows:

$$\underline{A} = \begin{bmatrix} A_r & 0 & 0 \\ A_{rg} & A_g & 0 \\ A_{rb} & A_{gb} & A_b \end{bmatrix} \quad \underline{B} = \begin{bmatrix} B_r & 0 & 0 \\ B_{gb} & B_g & 0 \\ B_{rb} & B_{gb} & B_b \end{bmatrix} \qquad (4a)$$

$$x = \begin{bmatrix} x_{rij} \\ x_{gij} \\ x_{bij} \end{bmatrix} \quad y = \begin{bmatrix} y_{rij} \\ y_{gij} \\ y_{bij} \end{bmatrix} \quad u = \begin{bmatrix} u_{rij} \\ u_{gij} \\ u_{bij} \end{bmatrix} \quad I = \begin{bmatrix} I_{rij} \\ I_{gij} \\ I_{bij} \end{bmatrix} \qquad (4b)$$

where subindexes *r,g,b* have been used to refer to color layers $\mathcal{L}_R$, $\mathcal{L}_G$, $\mathcal{L}_B$, respectively. Notice that although the state variables are independent of each other, layer interaction is permitted through the A and B templates, see Fig. 3. For instance, template A_{rg} has effect on both red and green layers simultaneously.

The characteristics generally used to distinguish one color from another are brightness, hue and saturation. Brightness embodies the chromatic notion of intensity. Hue is an attribute associated with the dominant wavelength in a mixture of light waves. Thus, hue represents a dominant color as perceived by the observer; when an object is called red, orange or yellow one is specifying its hue. Saturation refers to the relative purity or the amount of white light mixed with a hue. The pure spectrum of colors is fully saturated. Colors such as pink (red and white) are less saturated, with the degree of saturation being inversely proportionals to the amount of white light added.

Recall now from equation (2) that the CNN's output function is binary. In other words, if the color is taken directly from the output function the color would be either fully saturated or black. Moreover, combining the three saturated colors <RGB> would yield only a small gamut of distinct colors. To be able to make use of a full range of hues, the color information is taken from the cell's state rather than from the output itself. Notice however that the cell's state, x, is not bounded, and while with fully saturated colors there is a straight mapping from CNN output values to color intensities, e.g. $\mathcal{C}$: $\{-1,1\} \rightarrow \{0, 255\}$, the problem here is more complex as the state x can take any value. In other words, we need to find a function capable of mapping all real numbers to the closed interval [0..255], e.g. $\mathcal{C} : \mathbb{R} \rightarrow \{0,1,2... 255\}$.

We investigated two color mapping schemes: a *continuos* mode and a *quantized* mode. The latter one is based on a linear mapping using the maximum and minimum layer colors as bounds to generate a discrete (quantized) range of colors. The general mapping is as follows

$$z = (G - 1)\left[\frac{(x - x_l)}{(x_u - x_l)}\right] \tag{5}$$

where z is the quantized color, x is the current cell's state, x_l is the minimum layer state value, x_u is the maximum layer state value and G the absolute quantized value obtained as 2^g, for $0 \leq < g \leq 8$, with g the number of bit levels. Recall from equation (2) that valid state values exclude the open interval range (–1,1). This precludes the applicability of the general mapping as we have essentially negative state values which could yield negative pixel values. Therefore, the state is mapped to a linear function in the following form:

$$z = \frac{J}{2} + \frac{J}{2g}\log_2\left\{(G - 1)\left[\frac{(x - 1)}{(x_u - 1)}\right]\right\} \qquad for\ x > 1 \tag{6a}$$

$$z = \frac{J}{2} - \frac{J}{2g}\log_2\left\{(G - 1)\left[\frac{(x + 1)}{(x_u + 1)}\right]\right\} \qquad for\ x < -1 \tag{6b}$$

where J is the absolute maximum color value, i.e. J = 255. Notice that this value is split at half the color range. This is an arbitrary cut–off which has given us good visual perceptive results. Fig. 4 shows the benchmark used to test the mapping technique. The template used for this example corresponds to a non–filtering application characterized by minimum feedback and high gain feed forward as follows

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 200 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad I = -1 \tag{7}$$

This template is applied to all three layers $\mathcal{L}_R$, $\mathcal{L}_G$, and $\mathcal{L}_B$. A high entry value in the B template ensures that the strength of the input pixel value remains unchanged. This is further balanced with I *bias* whose negative value brightens every single pixel in the whole image. We found no color difference between the benchmark and processed result.

For the continuos mode mapping, the bilinear transformation employed in analog to digital filter transformations was applied. This transformation can be characterized by the function $C : R \rightarrow (-1,1)$. In other words, the set of real numbers is mapped to numbers bounded between –1 and 1. A second mapping can then be applied to the bounded numbers so that their values can be scaled to the appropriate color range values between 0 and 255. The general form of the bilinear transformation is given by

$$z = J\frac{x+1}{1-x} \tag{8}$$

where J is the maximum absolute color value and x is the state value. Notice that this particular function will map all states $x < -1$ to $|z| < 1$. This obviously leaves positive states unbounded. To overcome this problem the mapping is split in two transformations as follows

$$z = \frac{J}{2} - \left\{\frac{J}{2}\left[\frac{\bar{x}+1}{1-\bar{x}}\right]\right\} \qquad for\ x > 1 \tag{9a}$$

$$z = \frac{J}{2} - \left\{\frac{J}{2}\left[\frac{x+1}{1-x}\right]\right\} \qquad for\ x < -1 \tag{9b}$$

where $\bar{x}$ is taken as the negative of the current state value. The mapping was also tested on the benchmark of Fig. 4. This technique presented a small error of 2.3% on each primary color. This difference is actually not perceived by simple visual inspection.

We can conclude that both color mapping techniques are good. From our own experience we have noticed that for applications in which sharp color changes need to be highlighted the quantized mode projects good visualization results. In applications for which the color change is smooth the continuos mode is very suitable. Examples of applications with sharp color changes are edge detection and thresholding, and applications with smooth color changes are averaging and noise removal, to just mention a few.

4. IMAGE BASED BEHAVIORAL SIMULATION

Recall that equation (1) is space invariant, which means that $A(i,j;k,l) = A(i–k,j–l)$ and $B(i,j;k,l) = B(i–k,j–l)$ for all i,j,k,l. Therefore, the solution of the system of difference equations can be seen as a convolution process between the image and the CNN processors. The basic approach is to imagine a square subimage area centered at (x,y), with the subimage being the same size of the templates involved in the simulation. The center of this subimage is then moved from pixel to pixel starting, say, at the top left corner and applying the A and B templates at each location (x,y) to solve the differential equation. This procedure is repeated for each time step, for all the pixels. An instance of this image scanning–processing is referred to as an "iteration". The processing stops when it is found that the states of all CNN processors have converged to steady–state values, and the outputs of its neighbor cells are saturated, e.g. they have a ±1 value. This whole simulating approach is referred to as *raster simulation*. The raster approach implies that each pixel is mapped onto a CNN processor. That is, we

have an image processing function in the spatial domain that can be expressed as $g(x,y) = T(f(x,y))$ where $f(\cdot)$ is the input image, $g(\cdot)$ the processed image, and T is an operator on $f(\cdot)$ defined over the neighborhood of (x,y). For CNN this means that an output image pixel is only influenced by input image pixels within some extent area r in the neighborhood of the corresponding output image pixel. In common image processing applications $T(\cdot)$ is usually carried out as a convolution process between a response function array and the input image. For CNN this is a 2–D convolutional layer with feedback. The layer extracts information from two maps, i.e. the input and the current output states, using templates A and B to produce new output states. For image processing applications it is convenient to restrict the input and output arrays to be of the same dimension. Notice that when the templates are located on the border of the input image, the convolution process of A and B does not involve all of its elements. To deal with this border effect a center zero padded superposition model is used. That is to say, a virtual set of border cells, initialized to zero state values, is created. The multi–layer CNN raster simulation algorithm is presented in Fig. 5.

For the purpose of solving the initial–value problem, well established single–step methods of numerical integration techniques are used[18]. Three of the most widely used single–step algorithms are applied in the CNN behavioral simulator described here. They are the Euler's algorithm, the Improved Euler Predictor–Corrector algorithm and the Fourth–Order (quartic) Runge–Kutta algorithm. Euler's method is the simplest of all algorithms for solving ODEs. It is an explicit formula which uses the Taylor–series expansion to calculate the approximation. The Improved Euler Predictor–Corrector method uses both explicit (predictor) and implicit (corrector) formulae. The integral is calculated by multiplying a step size τ with the averaged sum of both the derivative of the discretized state, $x(n\tau)$, and the derivative of the predicted state, $x_p((n+1)\tau)$, at the next time step. The 4th order Runge–Kutta method is the most costly among the three methods in terms of computation time, as it requires four derivative evaluations per time step. However, its high cost is compensated by its accuracy in transient behavior analysis.

Since speed is one of the main concerns in the simulation, finding the maximum step size that still yields convergence for a template can be helpful in speeding up the system. The speed–up can be achieved by selecting an appropriate step size τ for that particular template, See Fig. 6a. The importance of selecting an appropriate τ can be easily visualized in Fig. 6b. If the step size chosen is too small, the simulation might take many iterations, hence longer time to achieve convergence. On the other hand, if the step size taken is too large, the simulation might not converge at all or it would converge to erroneous steady state values; the latter remark can be observed for the Euler integration method in the plots of Fig. 6b. The past results were obtained by simulating a small image of size 16x16 (256 pixels) using an Edge Detection template on a diamond figure on only one layer. In Fig. 7, simulation time computations using an averaging template for images of sizes to about 250,000 pixels are shown.

5. A CNN POST–PROCESSOR

The simulator was built using many of the features of the public domain software XPaint[19]. The environment is menu driven and allows to create color palletes, new canvas, in addition to the standard graphics features such as brushes, lines, circles, etc.

The intrinsic features of CNN were grouped under one single menu. Among the features of the CNN software are capabilities for single layer, multi–layer and time multiplexing simulations, control of the numerical integration, control over the initial conditions of the process and the color mode in which the system operates. Templates can also be edited on–line making out of this a very useful feature especially during the development of new applications.

Doing image processing with CNN may not always yield the desired results and post–processing becomes then necessary. The CNN post–processor consists of a compiler capable of handling logical pixelwise operations among distinct color layers. This compiler follows the trends of having CNN as an analogic microprocessor[20]. The added capability allows to create new processed images with say, one layer processed by CNN and the remaining layers logically manipulated between CNN results and the original image. Detailed examples of this extended processing capability are given in the next section. Fig. 8 shows the syntax of the post–processing language using a Backus Naur Form notation; keywords and variables are identified as boldface and italic words, respectively.

All the files to be processed by this CNN pot–processor must be specified at the beginning of the program as indicated in statement 1. When a file is read, the program splits the pixel information into its basic *RGB* components. This strategy is used to create three unique layers that contain the color coded information of the image. Statement 3 shows the logical pixelwise operations among layers. These operations include the conventional NOT, OR, AND, XOR, shift–left, and shift–right functions indicated in statement 6. Operations can be performed on the layers of a file or a variable but must always be stored in a variable. The only three valid layers are assigned to the triplet *<RGB>* and are specified by means of keywords, see statement 8. Every variable's layer is initialized to "black" when first used. Finally, the new processed image is spooled out in statement 9 in which it is required to specify the name of the output file and the variable containing the image to be printed.

6. COLOR IMAGE PROCESSING USING CNN

This section of the paper will try to demonstrate the capabilities of our software and the enormous potential that CNN has on a wide variety of applications. For this purpose, we will present five examples with applications in medical image processing, weather forecast, image restoration and simple color manipulation.

The first example deals with color contrasting. Let us bring your attention to the top region of Fig. 9a (160,590 pixels) at the height of the austronaut's helmet. Here, it is very difficult to perceive a set of clouds hidden in the blue background. This image was processed with the following templates[21]

$$A = \begin{bmatrix} 0.01 & -0.075 & 0.01 \\ -0.075 & 1.28 & -0.075 \\ 0.01 & -0.075 & 0.01 \end{bmatrix} \quad B = \begin{bmatrix} -0.04 & -0.13 & -0.04 \\ -0.12 & 0.71 & -0.13 \\ -0.04 & -0.13 & -0.04 \end{bmatrix} \quad I = -0.365 \qquad (10)$$

The purpose of this template combination is to do a soft edge detection on all three color layers. The simulator was set to operate in a quantized color mode and was stopped after 3 iterations. The resulting process image with the clouds uncovered is shown in Fig. 9b. Fig. 9c shows the same image after the edge detection process was completed in 21 iterations. Edging and contrasting operations performed by CNN are quite obvious. This particular example raises the following interesting remark. Notice that although CNN is searching for the steady state solution of a partial differential equation, in image processing applications intermediate or partial solutions may be sufficient to visualize the results.

The second example deals with the X–rays image of a chest cage (148,370 pixels) displayed in Fig. 10a. The objective is to color–code the black and white image and to highlight hidden features in the esternun. The image was treated two times with distinct templates. First, a "pixel peeler" template was used to widen visual and hidden contours in the image. The resulting image is displayed in Fig. 10b. Observe the black dots along the contour of the ribs. This template was chosen instead of a common edge detector because the latter leaves only the contours and darkens the body of the image. This would actually alter the information contained in the image as our goal was to only highlight the edges. The template "peels" the rightmost pixel from any two or more adjacent pixels; this action is executed through the *B* template. Both templates are as follows

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 3 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad I = -1 \qquad (11a)$$

The resulting image was further processed by a "filler template". The template was applied only to the red layer; the other two remaining layers were processed with the unity template described by equation (7). These templates are as follows

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad I = -1 \qquad (11b)$$

Basically, the function of the *A* template is to feed the pixel values of the neighboring pixels into the current pixel. A negative bias is used to avoid having an excessively dark image. The result of this operation is shown in Fig. 10c. The CNN processor undoubtedly did its work. Unfortunately, the visualization of the results is not optimal as the image has still many gray tones. Hence, the image was post–processed to achieve an optimal color manipulation. The post–processing program is listed below

```
(main original.gif edge.gif edge2.gif
(xx->red, edge.gif->red || edge2.gif->red)
```

```
(xx->green, original.gif->green || edge.gif->green)
(xx->blue, original.gif->blue)
(output (out,xx)) )
```

The files `original.gif`, `edge.gif`, and `edge2.gif` correspond to the original black and white image, the image processed in the continuos color mode, and the image processed in the quantized mode, respectively. The program does the following. *i)* The red layer of both edge detection operations is ORed to obtain soft and hard tone contours, *ii)* the green layer of the original image is ORed with the green layer of the edge detection obtained using the quantized color mode, and *iii)* the blue layer is left intact. The result of this post processing is shown in Fig. 10d. It is quite obvious that CNN was able to detect the hidden features in the esternun which otherwise are impossible to perceive from the original black and white image.

Our next example concerns image restoration. Fig. 11a displays the famous Monalisa (196,312 pixels) with a white scratch along the eyes. The restored image is displayed in Fig. 11b. This particular image is very difficult to restore because the background in the neighborhood of the scratch is not made of a solid color. The background presents a very irregular and granular surface which cannot be restored using only and Average Template. The use of such a template would tend to smooth out the granular structures leading to an incorrect restoration. Instead, a sequence of vertical and horizontal pixel peeling templates in addition to the standard averaging template were used.

This forthcoming example demonstrates CNN's ability to color–code a black and white image. The image presented in Fig. 12a corresponds to an actual satellite weather map (200,984 pixels) taken on May 25 1994 (the reader can see that we had a very cloudy weather). The goal was to obtain a result as close as possible to the standard color–coded maps used in weather forecasts. These maps use bright tones of green to show light clouds up to red tones to show strong rain intensities. The image was treated with color thresholding techniques using intensively the bias factor I. The result of this operation is displayed in Fig. 12b. The templates are as follows

$$A_{red} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2.9 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad A_{green} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 20.2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad A_{blue} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad I_{red} = -0.25 \quad I_{green} = -5.9 \quad I_{blue} = -0.55$$

The *A* template is used basically to operate on the quality of the colors. A large value in the center of this template helps to obtain a brighter color on the result. This explains the very high value for the green layer. A large value in the center of the *B* template is used to reinforce the effect of the input image on the final result. The thresholding function of *I bias* works as follows. Negative values, force the color of the layer to dominate on the final image. In other words, if we make *I bias* very

negative, a weaker degree of gray will be enough to make the corresponding color appear on the resulting image. This explains the different values adopted for *I*. The result was further processed by the following program

```
(main map.gif
(w->red, map.gif->blue || map.gif->red)
(w->green, map.gif->green ^ map.gif->blue)
(output (out,w)))
```

This program changes the white color into red, allowing us to obtain the result that we were looking for, see Fig. 12c.

Thelast example is again in the area of medical image processing. In this case CNN is used to highlight the extraction of blood vessels in a cross section from a cardiac image (68,040 pixels), see Fig. 13a. The procedure consisted in applying a strong edge detection template with the simulator working in the quantized color mode. The templates used in this operations look as follows

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} -0.25 & -0.25 & -0.25 \\ -0.25 & 2 & -0.25 \\ -0.25 & -0.25 & -0.25 \end{bmatrix} \quad I = 3.5 \qquad (12)$$

The *B* template resembles more the Laplacian operation commonly performed in digital image processing operations[7]. A large value in the *A* template reinforces only the output pixel value. A positive *I bias* leaves the areas outside the detected contour in a dark color. The result of applying these templates is displayed in Fig. 13b. Notice that the contour is perfectly detected. The post processor was used with this image together with the original image. The program used is as follows

```
(main edge.gif original.gif
(xx->red, ~edge.gif->red)
(xx->red, xx->red && original.gif->red)
(xx->green, original.gif->green)
(xx->blue, edge.gif->blue ^ original.gif->blue)
(output (out,xx)))
```

The files `edge.gif` and `original.gif` contain the processed and original images, respectively. The post processing objective is to delete the red color from the white areas of Fig. 13b and to highlight them in a color–coded fashion as these are the areas that contain the blood vessels. The blood vessels will be highlighted using green and blue tones. To do this, the red color layer is inverted to delete the color from the edge detected white area and the green and blue layers are processed with the original image. The result of this operation is shown in Fig. 13c.

7. CONCLUSION

A Cellular Neural Network environment for color image processing was presented in this paper. The work hereby introduced advanced the state of the art into processing of color images. It was demon-

strated that by collecting results from the cell's state rather than from its soft limited output, it is possible to obtain a full gamut of color tones. It was observed that this mapping of cell states to color values is not straightforward because the states are not bounded. To overcome this limitation two color mapping schemes were introduced that effectively assign states to distinct color hues. The error produced by these schemes is minimum. Therefore they are deemed to be very suitable for CNN color simulations.

Finally, from the examples presented in the last section, one can see the unquestionable potential of CNN in image processing applications.

ACKNOWLEDGEMENT

The author thank Mr. Maurizio Basso for developing the templates to process the weather map image.

REFERENCES

[1] L.O. Chua and T. Roska, "The CNN Paradigm," *IEEE Trans. on Circuits and Systems –1: Fundamental Theory and Applications,* vol 40, no. 3, pp. 147–156, March 1993.

[2] T. Matsumoto, T. Yokohama, H. Suzuki, and R. Furukawa, "Several image processing examples by CNN, " *Proc. IEEE Int. Workshop on Cellular Neural Networks and Their Applications,* pp100–11, 1990.

[3] T. Matsumoto, L.O. Chua, and H. Suzuki, "CNN clonning template: Shadow detector, " *IEEE Trans. Circuits Syst.*, vol 37, pp. 1070–1073, Aug. 1990.

[4] T. Matsumoto, L.O. Chua, and H. Suzuki, "CNN clonning template: Connected Component Detector, " *IEEE Trans. Circuits Syst.*, vol 37, pp. 633–635, May. 1990.

[5] T. Matsumoto, L.O. Chua and T. Yokohama, "Image Thining with a Cellular Neural Network," *IEEE Trans. on Cricuits Syst.*, vol 37, pp. 633–635, May 1990.

[6] K.R. Crounse, T. Roska, and L.O. Chua, "Image Halftoning with Cellular Neural Networks," *IEEE Trans. Ciruits & Syst.*, vol. 40, pp. 267–283, April 1993.

[7] T. Roska, A. Zarándy, L.O. Chua, "Color Image Processing using Multi–layer CNN Structure, " *H Didiev (ed), Circuit Theory and Design 93,* Elsevier, Amsterdam, 1993.

[8] L.O. Chua and B.E. Shi, "Multiple Layer Cellular Neural Networks: A tutorial," *Algorithms and Parallel VLSI Architectures*, E.F. Deprettere and A. van der Veen, Eds., vol. A: Tutorials, New York: Elsevier, 1991, pp. 137–168.

[9] C.C. Lee and J. Pineda de Gyvez, "Single–Layer CNN Simulator," *Proc. IEEE Int. Symposium on Circuits and Syst.*, 1994

[10] C.C. Lee and J. Pineda de Gyvez, "Time–Multiplexing CNN Simulator", *Proc. IEEE Int. Symposium on Circuits and Syst.*, 1994

[11] L. Fausett, "Fundamentals of Neural Networks, Architectures, Algorithms and Applications," *Prentice Hall,* Englewood Cliffs, 1994

[12] L. O. Chua and L. Yang, "Cellular Neural Networks: Theory," *IEEE Trans. Circuits and Systems*, Vol. CAS–35, pp. 1257–1272, 1988.

[13] L. O. Chua and L. Yang, "Cellular Neural Networks: Applications," *IEEE Trans. Circuits and Systems*, Vol. CAS–35, pp. 1273–1290, 1988.

[14] K. Slot, Determination of cellular neural network parameters for feature dete of two dimensional images," *Proc. IEEE Int. Workshop on Cellular Neural Networks and Their Applications*, pp. 82–91, 1990.

[15] L.O. Chua and P. Thiran, "An anlytic method for designing simple cellular neural networks," *IEEE Trans. Circuits & Systs., vol 38, pp. 1332–1341, 1991.*

[16] W. K. Pratt, "Digital Imge Processing, 2nd edition," *John Wiley &Sons, Inc.*, New York, 1991

[17] R.C. Gonzales and R.E. Woods, "Digital Image Processing,"*Addison Wesley Publishing Co.*, Reading Massachusetts, 1992

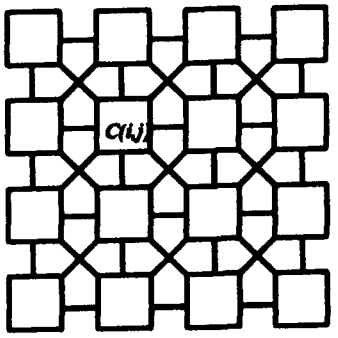
[18] W. H. Press, B. P. Flannery, S.A. Teukolsky, and W.T. Vetterling, "Numerical Recipes. The Art of Scientific Computing", Cambridge University Press, New York, 1986

[19] D. Koblas, "XPaint," public domain software.

[20] L.O. Chua and T. Roska, "The CNN Universal Machine Part 1: The Architecture", in *Int. Workshop on Cellular Neural Networks and their Applications (CNNA)*, pp. 1–10, 1992.

[21] F. Zou, S. Schwarz, and J. A. Nossek, "Cellular Neural Network Design Using a Learning Algorithm," *Proc. IEEE Int. Workshop on Cellular Neural Networks and their Applications,* pp. 73–81, 1990.

LIST OF FIGURES



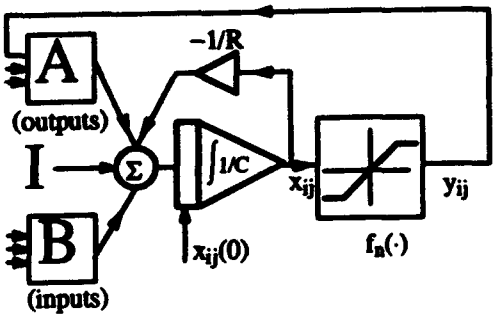


Fig. 1

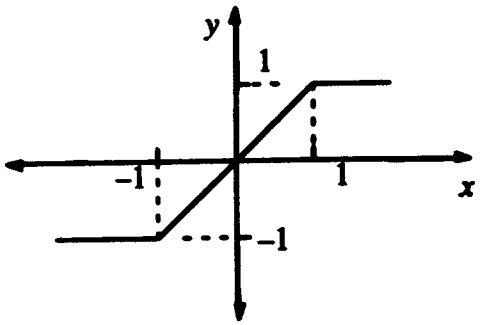


Fig. 2

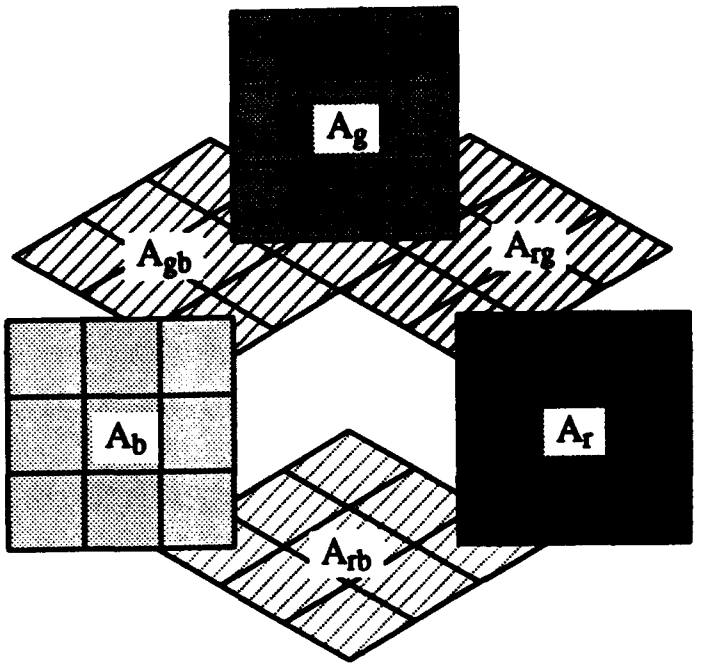


Fig. 3

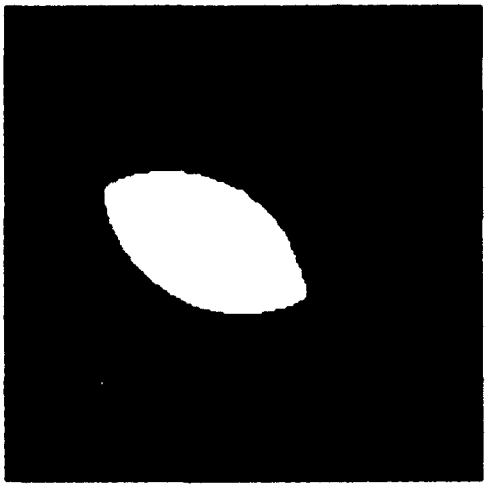


Fig. 4

Algorithm: (Multi-Layer Raster CNN simulation)

Obtain the input image, initial conditions and templates from user;

/* M,N = # of rows/columns of the image */

while (converged_cells < total # of cells) {

  for (layer=0; layer < 3; layer++) {

    for (i=1; i<=M; i++)

      for (j=1; j<=N; j++) {

        if (convergence_flag[layer][i][j])

          continue; /* current cell already converged */

        /* calculation of the next state*/

        $$x_{layer,ij}(t_{n+1}) = x_{ij}(t_n) + \int_{t_n}^{t_{n+1}} f(x(t_n))\, dt$$

        /* convergence criteria */

        if ($\frac{dx_{layer,ij}(t_n)}{dt} = 0$ and $y_{layer,kl} = \pm 1,\ \forall\ C_{layer}(k,l) \in N_r(i,j)$) {

          convergence_flag[i][j] = 1;

          converged_cells++ ;

          }

      } /* end for */

    /* update the state values of the whole image*/

    for (i=1; i<=M; i++)

      for (j=1; j<=N; j++) {

        if (convergence_flag[layer][i][j]) continue;

        $x_{layer,ij}(t_n) = x_{layer,ij}(t_{n+1})$;

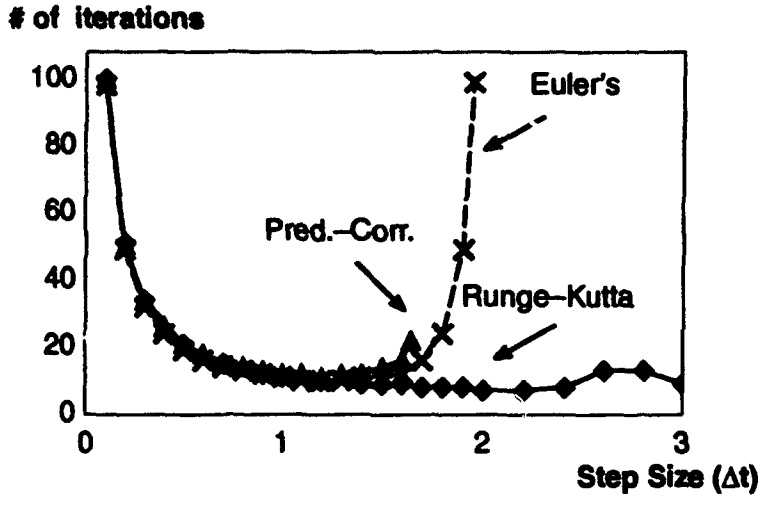
      }

    #_of_iteration++;

  }

} /* end while */

Fig. 5


# of iterations
100
80
60
40
20
0
0
1
2
3
Step Size (Δt)
Euler's
Pred.–Corr.
Runge–Kutta

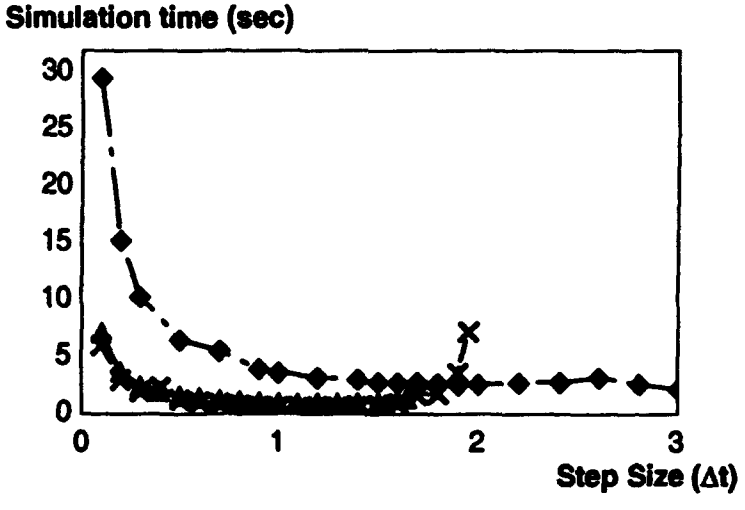

Simulation time (sec)
30
25
20
15
10
5
0
0
1
2
3
Step Size (Δt)

Fig. 6

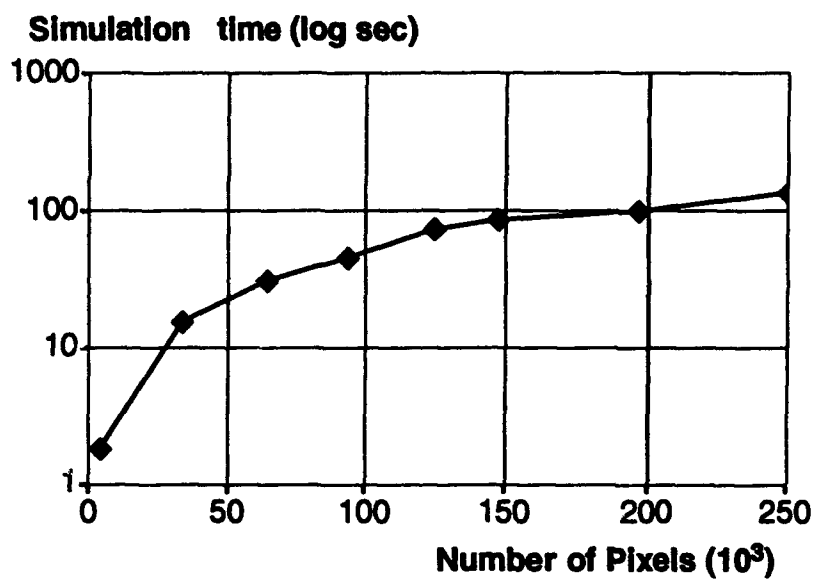

Simulation time (log sec)
1000
100
10
1
0
50
100
150
200
250
Number of Pixels (10^3)

Fig. 7

```
1: main_file      ::= "(" MAIN (files)+ (process)+ (output)+ ")"  .
2: files          ::= NAME .
3: process        ::= "(" (process_descr) ")"  .
4: process_descr  ::= (var) "->" (layer) "," (var) "->" (layer)
                        (operand) (var) "->" layer .
                  |   (var) "->" (layer) "," (var) "->" (layer) .
                  |   (var) "->" (layer) "," (var) "->" (layer)
                        (operand) NUMBER.
                  |   (var) "->" (layer) "," NUMBER (operand) (var)
                      "-" ">" (layer).
                  |   (var) "->" (layer) "," (negation) (var)
"->"                        (layer) .
5: var            ::= NAME .
6: operand        ::= (AND | "&&") .
                  |   (OR  | "||") .
                  |   (XOR | "^") .
                  |   (SL  | "<<") .
                  |   (SR  | ">>") .
7: negation       ::= NOT .
8: layer          ::= RED .
                  |   GREEN  .
                  |   BLUE   .
9: output         ::= "(" OUTPUT "(" (files) "," (var) ")" ")" .
```

Fig. 8

Fig. 9 (a)

Fig. 9 (b)

Fig. 9 (c)

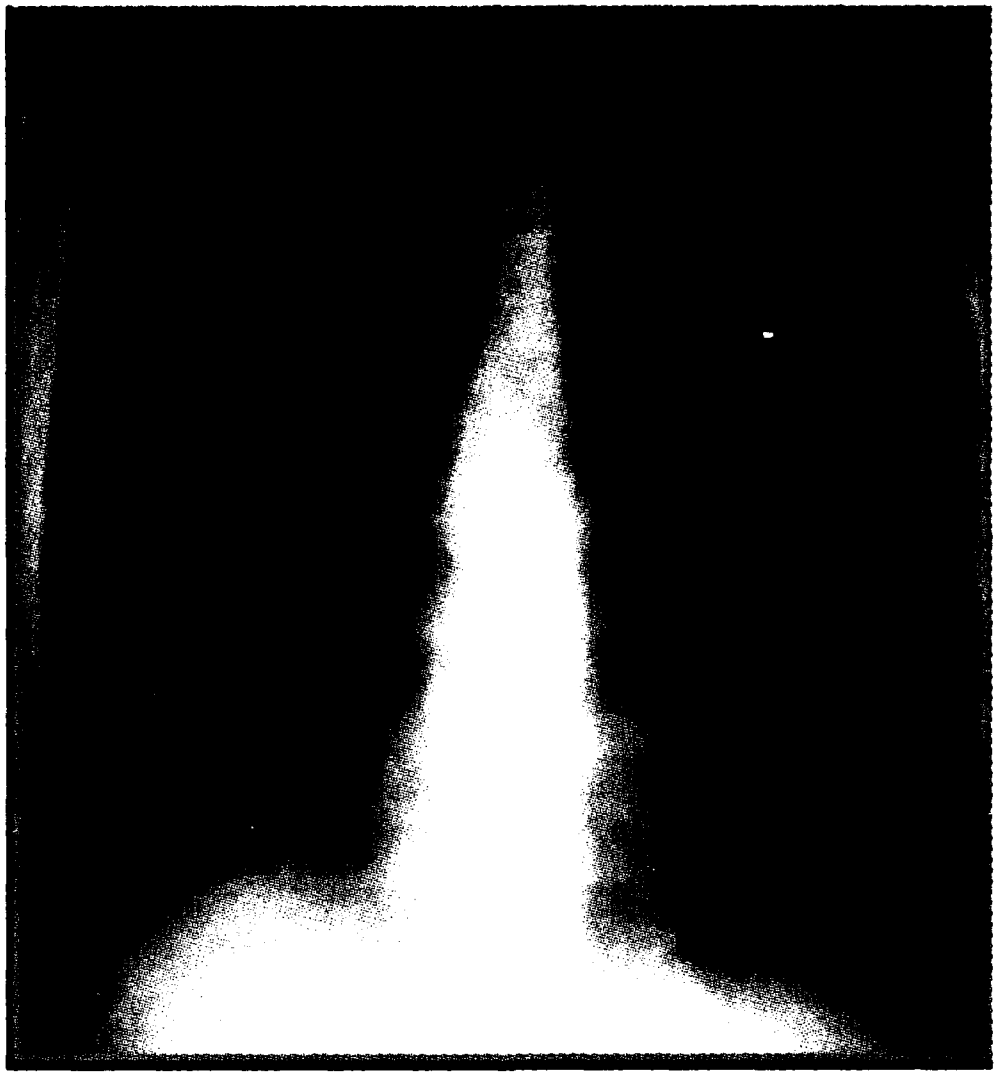


Fig. 10 (a)

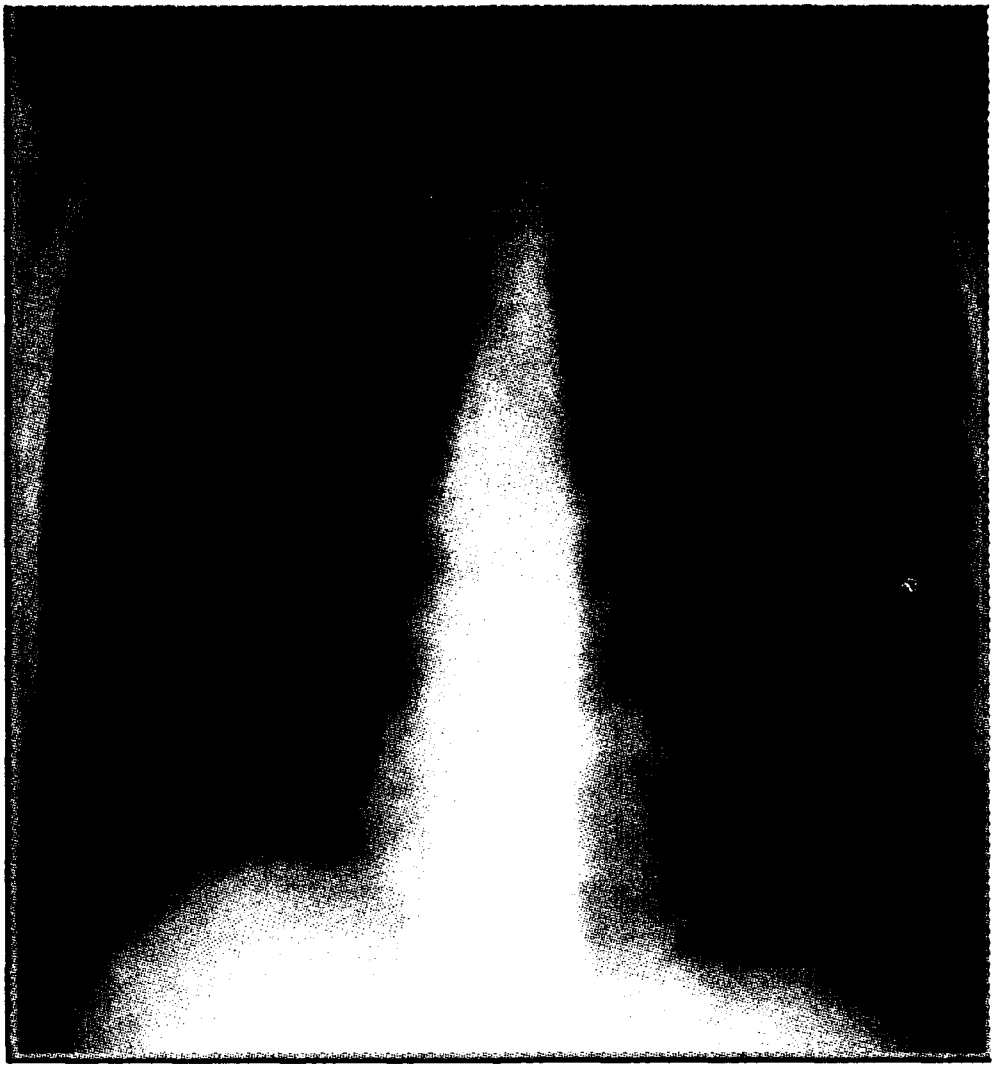


Fig. 10 (b)

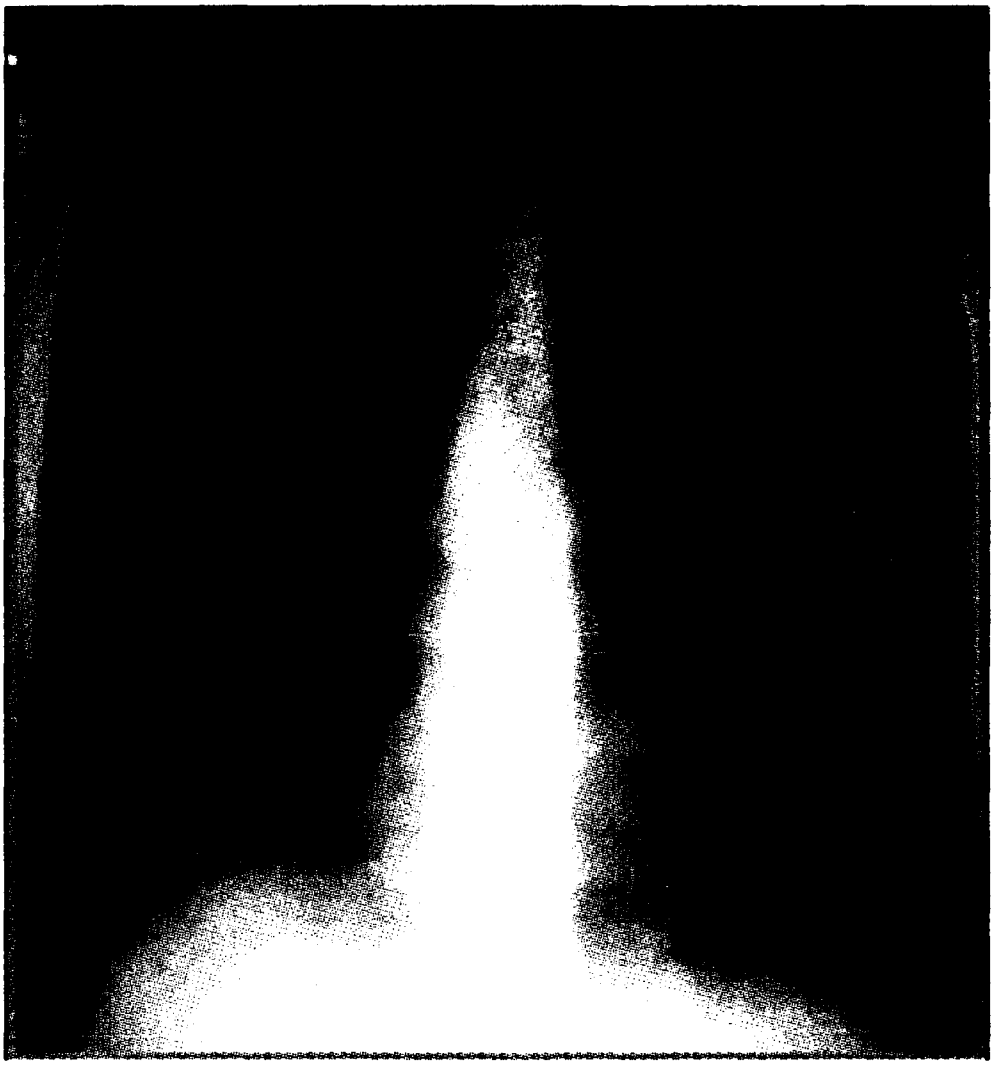


Fig. 10 (c)

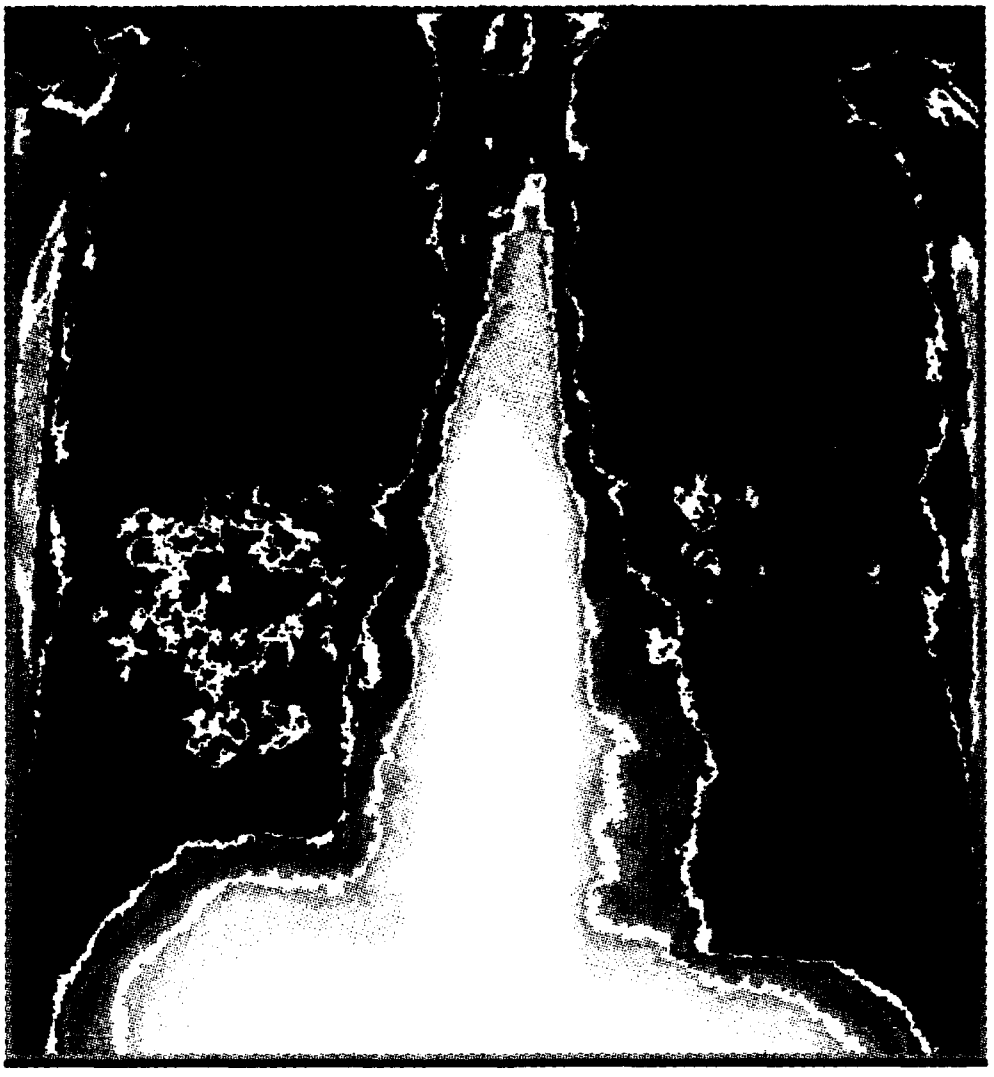


Fig. 10 (d)

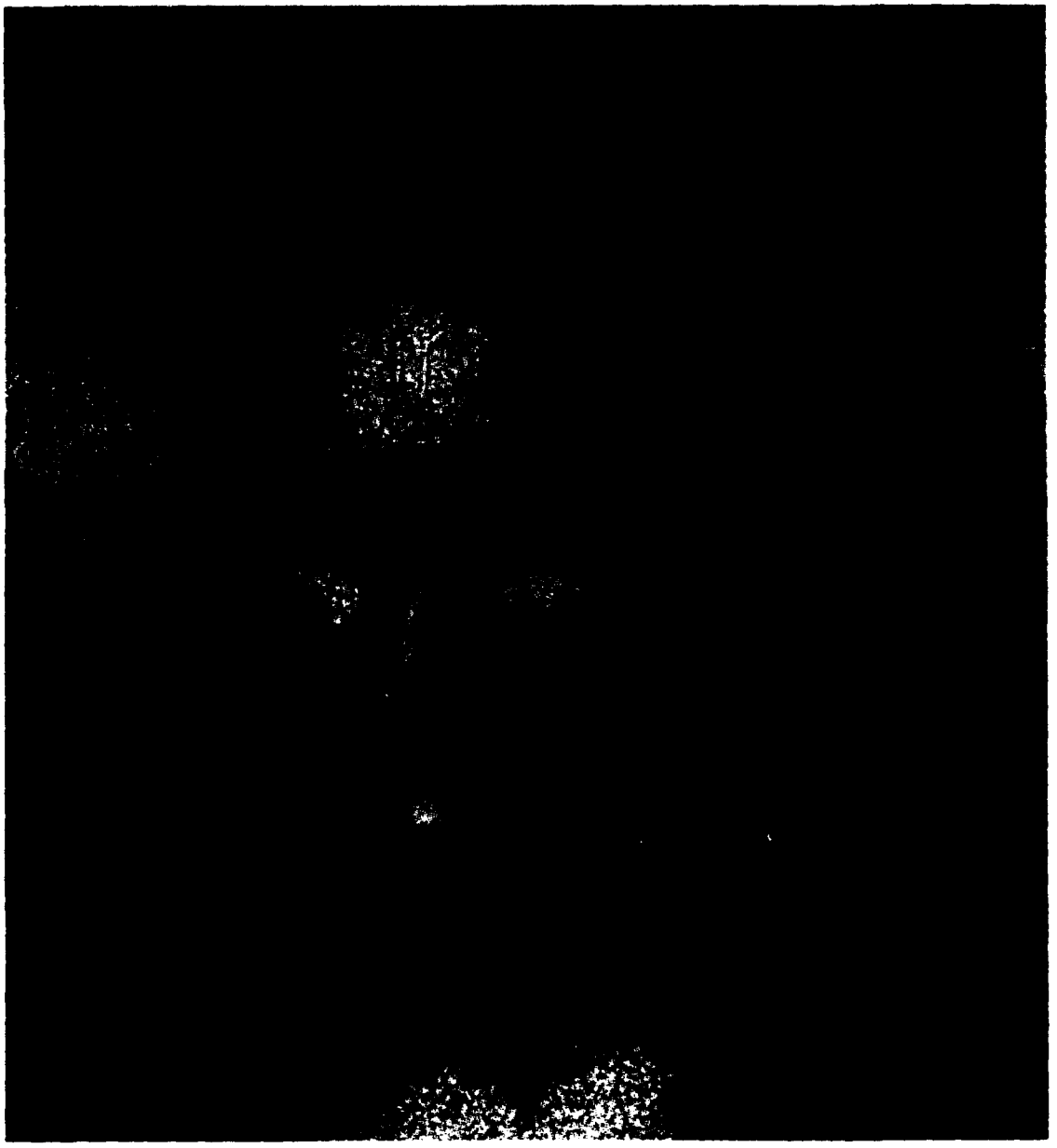


Fig. 11 (a)

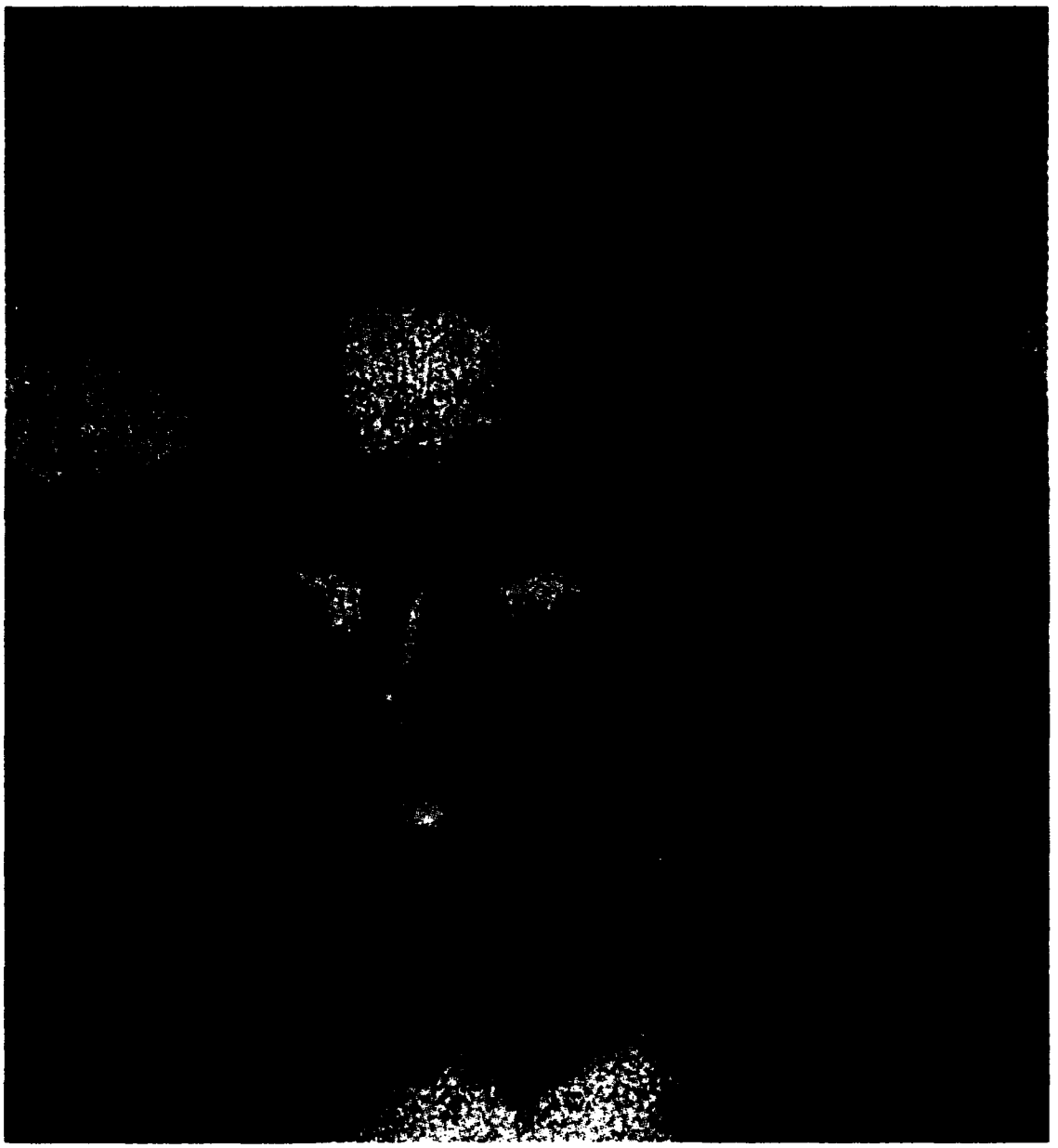


Fig. 11 (b)

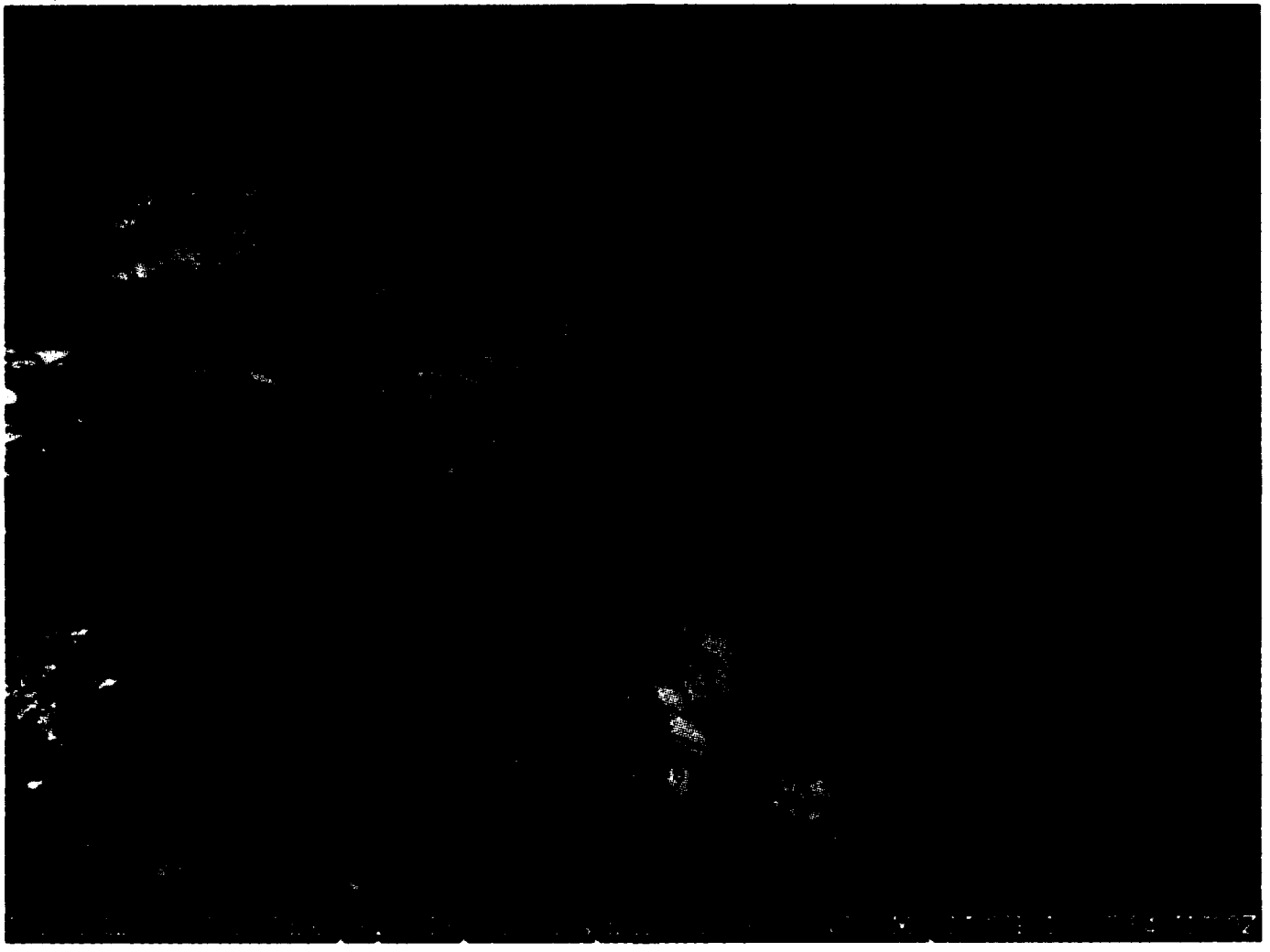


Fig. 12 (a)

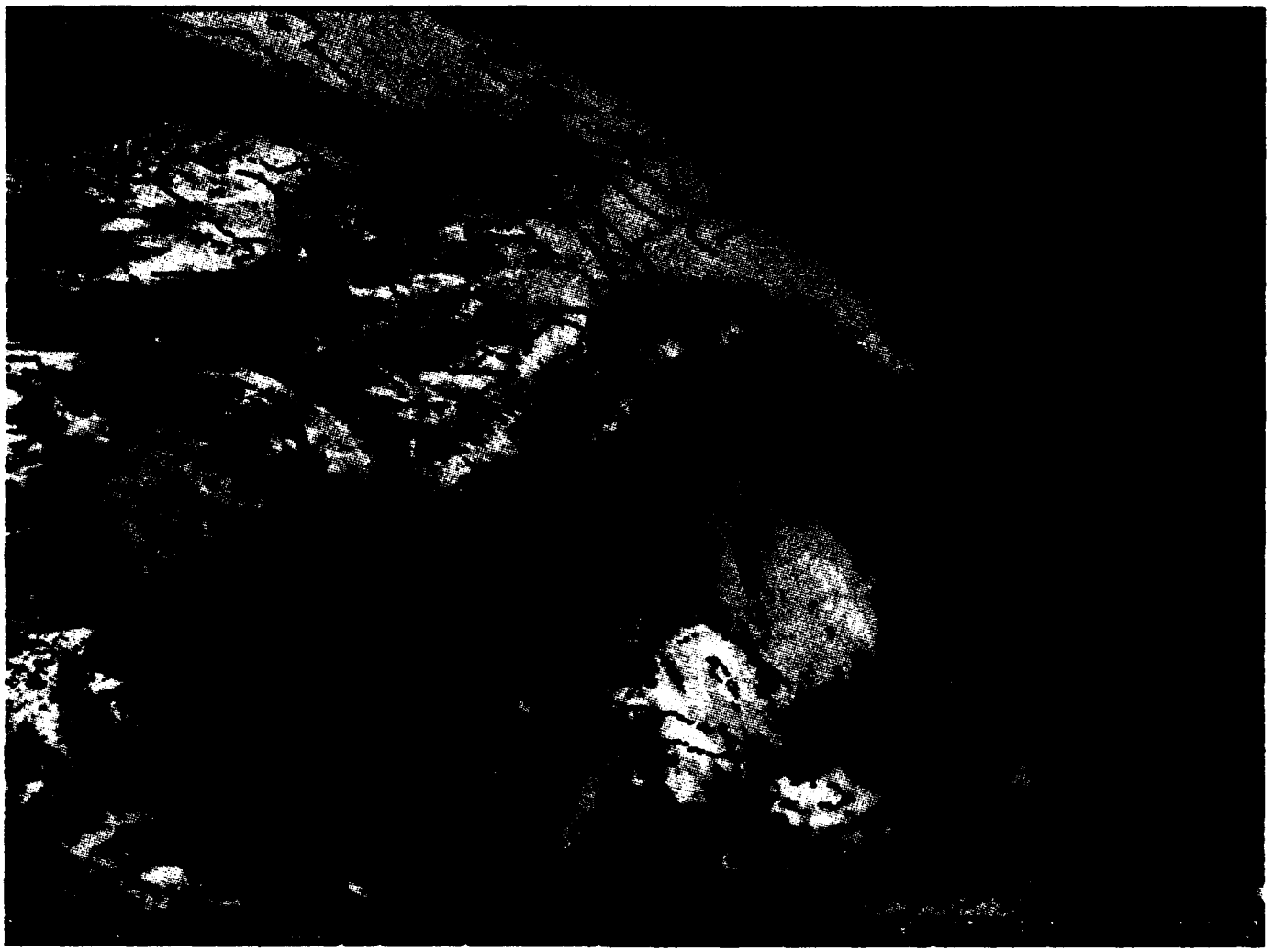


Fig. 12 (b)

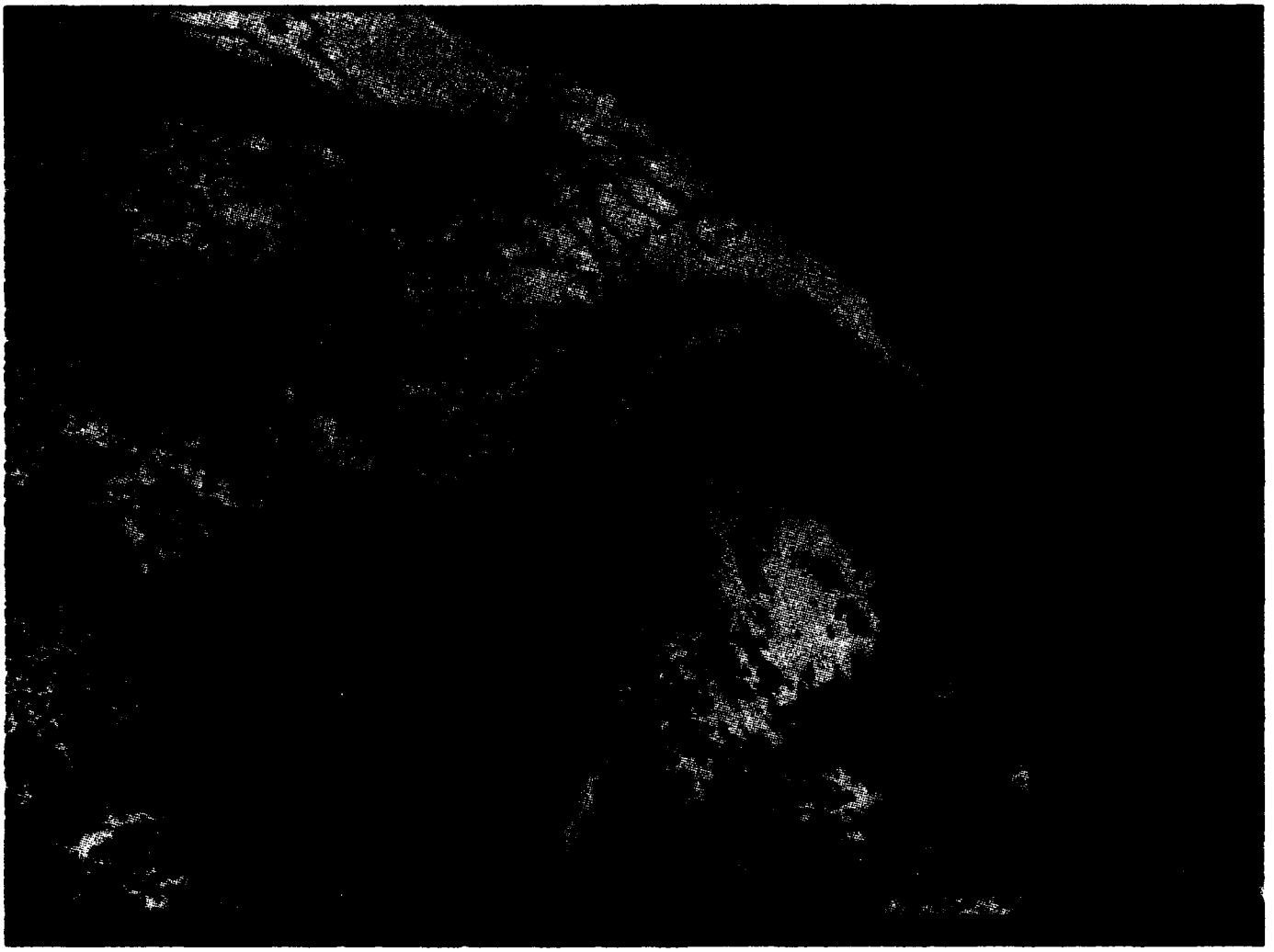


Fig. 12 (c)

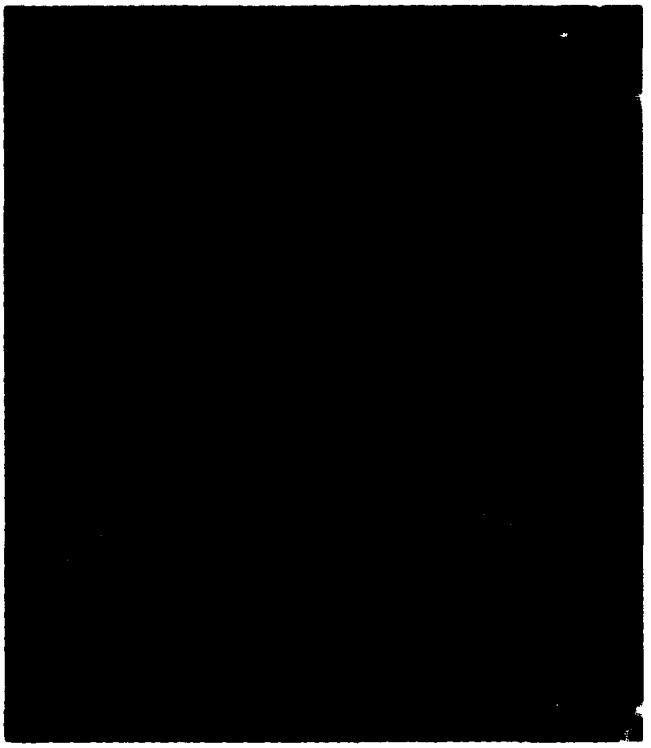


Fig. 13 (a)

Fig. 13 (b)

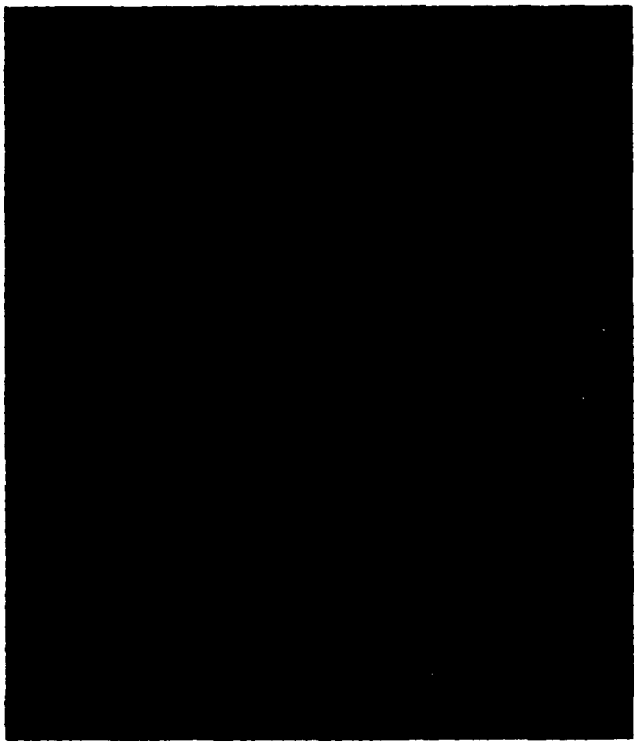


Fig. 13 (c)

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 3, 1994	Technical Report 05/01/94-08/01/94

4. TITLE AND SUBTITLE
COLOR IMAGE PROCESSING USING CELLULAR NEURAL NETWORKS

5. FUNDING NUMBERS
G
N00014-94-1-0516

6. AUTHOR(S)
Jose Pineda de Gyvez

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Texas Engineering Experiment Station
Texas A&M University

8. PERFORMING ORGANIZATION REPORT NUMBER
93-549/#2

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research
Code 215: JWK
Ballston Tower One
800 North Quincy Street
Arlington, Virginia 22217-5660

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Results submitted for publication to IEEE Trans. on Neural Networks

12a. DISTRIBUTION/AVAILABILITY STATEMENT
A: Approved for public release: distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This report presents a software prototype capable of performing image processing applications using Cellular Neural Networks (CNN). The software is based on a CNN multi-layer structure in which each primary color is assigned to a unique layer. This allows an added flexibility as different processing applications can be performed in parallel. To be able to handle a full range of color tones, two novel color mapping schemes were derived. In the proposed schemes the color information is obtained from the cell's state rather than from its output. This modification is necessary because CNN has binary outputs from which only either a fully saturated or a black color can be obtained. Additionally, a post processor capable of performing pixelwise logical operations among color layers was developed to enhance the results obtained from CNN. Examples in the areas of medical image processing, image restoration and weather forecasting are provided to demonstrate the robustness of the software and the vast potential of CNN.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Color Image Processing using Cellular Neural Networks	40
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements***.

Block 1. Agency Use Only *(Leave blank)*.

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	PR	- Project
G	- Grant	TA	- Task
PE	- Program Element	WU	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

- DOD - See DoDD 5230.24, "Distribution Statements on Technical Documents."
- DOE - See authorities.
- NASA - See Handbook NHB 2200.2.
- NTIS - Leave blank.

Block 12b. Distribution Code.

- DOD - Leave blank.
- DOE - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
- NASA - Leave blank.
- NTIS - Leave blank.

Block 13. Abstract. Include a brief *(Maximum 200 words)* factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code *(NTIS only)*.

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.